HowTo Build a Diskless Debian Cluster

Markus Rosenstihl

Contents

1 HowTo Build a Diskless Debian Cluster

PDF Version of this document.

1.1 Objective

We want to build a *diskless* Debian cluster for high performance computing (HPC). The main purpose of the cluster are molecular dynamic simulations using GROMACS.

Our current cluster consists of:

1. one master node/head node
2. about 20 compute nodes *without* any hard disk drives:

a) 16 dual dodeca(12) core AMD Opteron CPUs with 16 GB RAM
b) 4 dual quad core AMD Opteron CPUs with 8 GB RAM, one of these is the master-node
c) 2 NAS with quad Xeon processors, one with 22 GB RAM and one with 12 GB RAM

3. shared `/home` directories will be provided by a NAS
4. shared `/data` directories will be provided by the other NAS. The master-node is connected via 20 GB/s InfiniBand (actually only 10 GB/s due to old mainboards)

We are, as of now (2013-12), still running our Cluster on Debian Lenny and we really need to upgrade to the current stable version, Wheezy. This new setup should work

with *native* Debian packages (with one exception allowed: `ganglia-webfrontend`), and ideally it should still work with the next debian release, Jessie.

This "HowTo" intends to explain the basic steps to get this cluster up and computing and includes a description about setting up the master node as well as how to create the nfs root for the diskless compute nodes. Nevertheless, this short document can not give all the background information which is maybe needed, but an effort is undertaken to explain the *why* for the critical parts.

Why do we do this manually anyway? Aren't there a number of tools to do that:

- perceus
- warewolf
- kestrelhpc
- pelicanhpc
- oneSIS
- and more ...

I found most of the tools either lack documentation, or are way to complex for our simple needs. Diskless nodes are sometimes to complicated to setup or not even possible. Some of the packages have a lot of external dependencies (mysql-server), whereas some are just outdated and do not offer Debian Wheezy packages (if they are packaged at all), others are end-of-live" and won't be supported anymore.

Debian-live would be another, actually rather appealing possibility, but for some reason the boot process never went past the `ipconfig` part. I think now that the reason was the second NIC interfering in the IP discovery process (see this bug report), which seems to be finally fixed after this setup was made to work.

Anyway, let's start building our diskless cluster!

1.2 Basic Master Node Setup

1.2.1 SSH Setup

SSH for the cluster nodes, less questions, less secure. This can be removed when the live image is finalized and the host key is not changing as often. The better alternative would be to change the image creation script and use a pre-existing host key (as iit is done now):

```
root@test-master-node:~#  cat /etc/ssh/ssh_config |grep -v ^#
Host 192.168.0.*
    StrictHostKeyChecking no
Host linux-*
    StrictHostKeyChecking no
```

1.2.2 DNSmasq

We use `dnsmasq`to provide host name resolution for the nodes. The nice side effect of this is that we also get a `tftp` server for network boot. The biggest advantage though is

the very simple configuration compared to isc-bind- and isc-dhcpd-server. For our two dozen of nodes `dnsmasq` is very sufficient.

Make sure that the daemon is enabled Check in `/etc/default/dnsmasq`:

```
ENABLED=1
```

Our basic configuration Complete DNSmasqd configuration file `/etc/dnsmasq.conf`. This is the file used during transition from the old master node to the new master node. The outbound NIC is `eth1` with the IP `10.0.0.161`:

```
domain-needed  # Never forward plain names (without a dot or domain part)
bogus-priv # Never forward addresses in the non-routed address spaces.
no-resolv # Do not get nameservers from /etc/resolv.conf
local=/cluster/ #queries in these domains are answered from /etc/hosts or DHCP only.
server=10.0.0.254@eth1 #  send queries to 10.0.0.254 via eth1
listen-address=192.168.0.254 # listen on address
addn-hosts=/etc/hosts_cluster # read host names from these files
addn-hosts=/etc/hosts_icluster
domain=cluster # domain for dnsmasq
dhcp-range=192.168.0.248,192.168.0.253,24h # we need one range for dhcp server to work
dhcp-ignore=tag:!known # Ignore any clients which are not specified in dhcp-host lines
dhcp-option=42,0.0.0.0 # the NTP time server address to be the same machine  as is running dnsmasq
dhcp-option=40,cluster # set the NIS domain name
dhcp-boot=pxelinux.0 # set the boot filename for netboot/PXE
enable-tftp
tftp-root=/srv/tftp
```

Configure the ip <—> host name assignment This is done in `/etc/dnsmasq.d/nodes`:

We want the nodes physical position to be easy to correspond to an sepcific ip, that means for exmaple high IP numbers at top of the rack. Skip this if you do not want this, a node can also be identified with ipmi identify (`ipmi-chassis -i10 -h ilinux-10` will turn it on for 10 seconds). A blinking LED will identify the node, on our nodes the light is blue and thus easy to distinguish of the other blinking LEDs, which are green and red.

```
# this file is read automatically,
# assigns IP and hostname to each node
dhcp-host=00:23:54:91:86:61,node-03,192.168.0.3,12h
dhcp-host=00:23:54:91:86:64,node-04,192.168.0.4,12h
dhcp-host=00:25:90:13:c3:96,node-05,192.168.0.5,12h
dhcp-host=00:25:90:13:c0:ce,node-06,192.168.0.6,12h
dhcp-host=00:25:90:13:c1:ba,node-07,192.168.0.7,12h
dhcp-host=00:25:90:12:84:60,node-08,192.168.0.8,12h
dhcp-host=00:25:90:57:48:70,node-09,192.168.0.9,12h
dhcp-host=00:25:90:57:48:b6,node-10,192.168.0.10,12h
```

```
dhcp-host=00:25:90:57:48:60,node-11,192.168.0.11,12h
dhcp-host=00:25:90:57:48:3a,node-12,192.168.0.12,12h
dhcp-host=00:25:90:57:46:ca,node-13,192.168.0.13,12h
dhcp-host=00:25:90:57:48:aa,node-14,192.168.0.14,12h
dhcp-host=00:25:90:57:48:ca,node-15,192.168.0.15,12h
dhcp-host=00:25:90:57:48:52,node-16,192.168.0.16,12h
dhcp-host=00:25:90:57:48:da,node-17,192.168.0.17,12h
dhcp-host=00:25:90:57:46:08,node-18,192.168.0.18,12h
dhcp-host=00:25:90:57:45:f8,node-19,192.168.0.19,12h
dhcp-host=00:25:90:57:48:ce,node-20,192.168.0.20,12h
```

(Another, similar file in the same directory serves the IP Adresses for IPMI network interfaces.)

- one way to get the MAC addresses is to switch a compute node on and check the arp table with `arp`. When a new MAC is found power on the next node, etc. This is how it is done with the *real* cluster tools like perceus, kestrel, warewulf, etc.
- another is read the MAC during boot up (tedious!)
- yet another one is the type the number from the delivery documents if available, i.e. burn-in test protocols.
- if you have an existing cluster try `arp -ni <private iface>`. This will get you a list of currently configured MAC and IP addresses.

Add the node names to `/etc/hosts` The dnsmasq daemon will use this to answer DNS requests. The entries can be created by script, i.e. like this:

```
for i in {1..20};do
    echo 192.168.0.$i linux-$(printf "%02i" $i) \
    linux-$(printf "%02i" $i).cluster
done
```

will create entries along the following patter which need to be appended to the `hosts` file (>> `/etc/hosts`).

```
192.168.0.1 linux-01.cluster linux-01
192.168.0.2 linux-02.cluster linux-02
```

Check the DNS resolver configuration The file `/etc/resolv.conf` contains among the domain and evtl. the domain-search-name a list of up to three nameservers. Depending on the configuration of the outbound NIC, this file could be overwritten by the DHCP client daemon. To make sure that the local DNS server, dnsamsq, is asked first one should modify the DHCP client configuration in the file `/etc/dhcp/dhclient.conf` and add the entry `prepend domain-name-servers 127.0.0.1;`. This insures that the DNS server on the localhost will always be asked first. If the outbound NIC is configured manually this is not necessary, the resolv.conf file will not change.

I recommend **not** to install `network-manager` as it could interfere with the manual configuration in `/etc/network/interfaces`!

1.2.3 NIS

There are a lot of available HowTos floating on the net describing the process of setting up NIS. Good ones are:

- Arch Linux
- FreeBSD Handbook, which is absolutely marvellous and, despite beeing BSD based, still useful for Linux!
- the original NIS HowTo
- Debian specific

Insure that the NIS domain is set correctly The NIS domain has to be the same for all the computers in the cluster accessed by the users.

- `dpkg-reconfigure nis`
- check/set with commands `ypdomainname` or `nisdomainname`
- check content of `/etc/defaultdomain`

Master node as master NIS server The master node will be the master NIS server for our cluster, one or both of the NAS could be set up as slave server to provide redundancy.

Check NIS server configuration in the file `/etc/default/nis`:
`NISSERVER=master # or slave or false`

Initialize yp database In order to initialize the yp database issue the following command: `ypinit -m`

If something does not work Sometimes it helps to reinitialize/update the servers yp maps. To that effect issue the commands in a terminal:

```
cd /var/yp; make all
```

You can modify the `/var/yp/Makefile` to suit your needs. As an example, you can only serve UIDs in a certain numerical range with the `MINUID` and `MAXUID` variables.

Define our subnet For security reasons define our subnet in `root@testserver:~# cat /etc/ypserv.securenets`. This will make sure that only requests from within our subnet will be answered:

```
# Always allow access for localhost
255.0.0.0    127.0.0.0
# This line gives access to our subnet 192.168.0.0/24
255.255.255.0       192.168.0.0
```

1.2.4 RSYSLOG

We want the compute nodes to log everything to the head node and nothing on the node itself, i.e. all log data will be forwarded to the head node. Otherwise, it could happen that the small RAM filessystem fills up due to logging.

The following configuraton makes the rsyslogd process listen for incoming log messages:

```
cat /etc/rsyslog.conf |grep -v -e ^# -e "^$
```

```
$ModLoad imuxsock # provides support for local system logging
$ModLoad imklog   # provides kernel logging support
$ModLoad immark  # provides --MARK-- message capability
$ModLoad imudp
$UDPServerRun 514
$ModLoad imtcp
$InputTCPServerRun 514
$ActionFileDefaultTemplate RSYSLOG_TraditionalFileFormat
... the rest left unchanged  ...
```

1.2.5 Ganglia

Ganglia is used to monitor the health of our cluster. It stores collected data in RRD files, for example network usage, uptime, load, etc.. Basically, ganglia consists of one ore more data collectors (`gmetad`) and multiple data sources (`gmond`). Installinge the Debian package:

```
apt-get install ganglia-monitor gmetad
```

Ganglia daemon `gmond` Delete the lines with `mcast` because we do not use/need broadcast addresses and replace the line in the `*_send_*` section with a `host` entry in `/etc/ganglia/gmond.conf`:

```
udp_send_channel {
  host = 192.168.0.254
  port = 8649
  ttl = 1
}
udp_recv_channel {
  port = 8649
}
tcp_accept_channel {
  port = 8649
}
```

Configuration of `gmetad` Change the trusted `data_source` for the `gmetad` collection daemon in `/etc/ganglia/gmetad.conf`:

```
# Allow `gmetad` to receive data on `localhost` and our internal IP
data_source "my cluster" localhost 192.168.0.254
```

1.2.6 Ganglia-Web

The ganglia `gmetad` daemon is now running and collects data in RRD archives. In order to visualize the data one needs the ganglia-web frontend. Installing it via `apt-get` leads to the installation of `apache2`. I wanted to use a smaller http server and chose `lighttpd` for this task. To this end one can also install the ganglia website directly from the sources, the dependencies need to be fulfilled manually.

To make the ganglia web interface work one needs to install the package `php5-cgi`.

Then, one has to enable `lighttpd` to execute php-cgi scripts. This is accomplished with the following links in the configuration directory `/etc/lighttpd/conf-enabled`:

```
root@test-master-node:/etc/lighttpd/conf-enabled# ls -lgG
total 0
lrwxrwxrwx 1 33 Dec 10 14:18 10-fastcgi.conf -> ../conf-available/10-fastcgi.conf
lrwxrwxrwx 1 37 Dec 10 14:19 15-fastcgi-php.conf -> ../conf-available/15-fastcgi-php.conf
```

Installing the ganlia web frontend itself fairly straight forward. Download the tarball from their SourceForge site and uncompress it. Then edit the `Makefile` to your needs:

```
# Location where gweb should be installed to (excluding conf, dwoo dirs).
GDESTDIR = /var/www/ganglia
# Gweb statedir (where conf dir and Dwoo templates dir are stored)
GWEB_STATEDIR = /var/lib/ganglia-web
# Gmetad rootdir (parent location of rrd folder)
GMETAD_ROOTDIR = /var/lib/ganglia
APACHE_USER =  www-data
```

Finally, execute `make install` and the files will be copied to the given directory. Make sure the owner and permissions are right, then go to `http://master-node/ganglia` and watch the status of your cluster, you should see some pretty graphs.

1.2.7 NFS Exports

We use `NFSv4` exports, i.e. do not forget the `fsid=0` paramter for the root `/srv` directory! Do not forget that NFS booting itself does **NOT** work with `NFSv4`, adjust the mount point ind the PXE configuration file accordingly, that means you have to use `host_ip:/srv/nfsrooot` instead of `host_ip:/nfsroot` at the kernel command line.

1.2.8 Postfix Mailserver

SLRUM can send status emails for jobs, we also want to send emails for ceratin events like high room temperature, failed harddrives, and similar stuff. As all of the cluster is *at least* in our private subnet there has to be a gateway for mails. Using the master node is obviously the first choice. Another point is, we want to be able to send these mails to **any** address, which means we need to relay outbound mail to our institutes mail server.

Here is short abstract about what we want our mail server to be able do:

- receive mails from anybody on out local private subnet

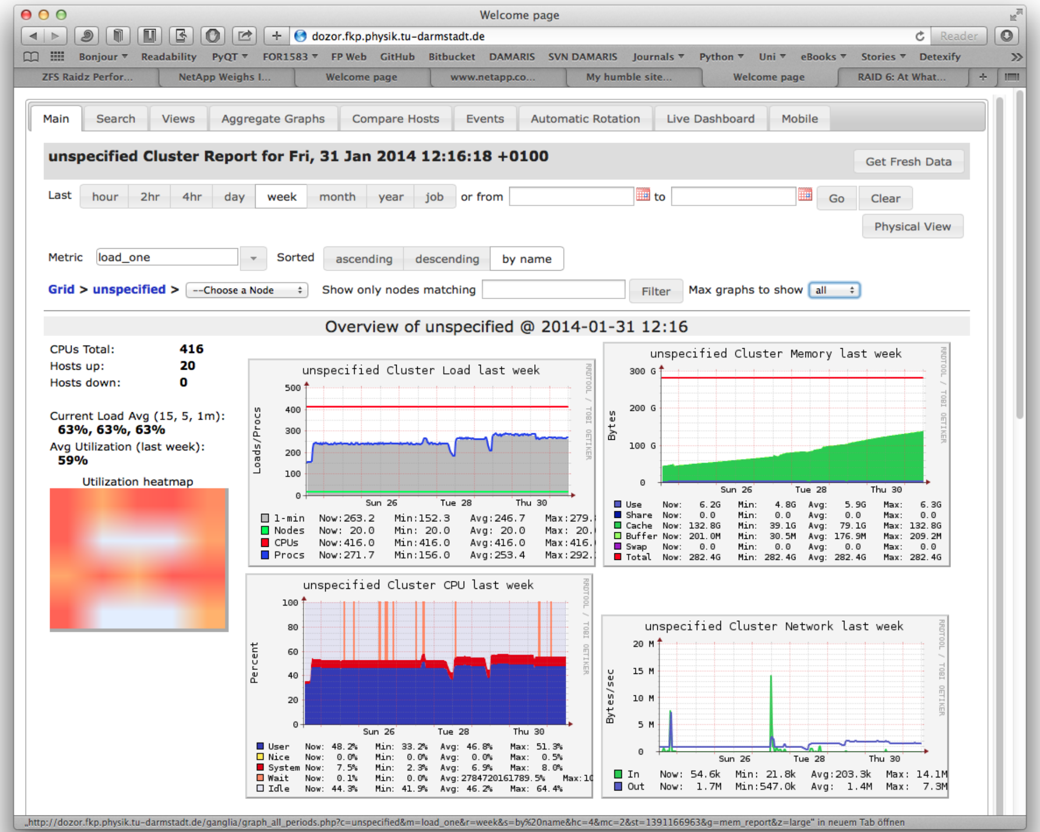

Figure 1: Ganglia cluster overview.

- send and relay emails to any outbound addresses
- do this *only* for mails originating from our local subnet (obviously)

Following are the important parts of our configuration in teh postfix configuration file `/etc/postfix/main.cf`:

```
myorigin = /etc/mailname
myhostname = master-node.cluster
mydestination = dozor.fkp.physik.tu-darmstadt.de, master-node, $myhostname,
    ↪ localhost.cluster-agvogel, localhost, localhost.localdomain
relayhost = relayhost_fqdn
mynetworks = 127.0.0.0/8 [::ffff:127.0.0.0]/104 [::1]/128 10.0.0.0/24
```

Short explanation:

- `myorigin` specifies the domain that appears in mail that is posted on this machine.
- `mydestination` lists the domains this machine will deliver locally. Do not forget to accept mail to `localhost`!
- `relayhost` will handle our non-local emails
- `mynetworks`: forward mail from clients in `$mynetworks` to any destination

A very detailed description about this configurationc an be found on the postfix website.

---

2 The Live Image for the Compute Nodes

2.1 Some general remarks

Following is an abbreviated description of the script used to build the NFS root for the diskless compute nodes. The most important steps are outlined here, the actual procedure is written in a script so that no step is forgotten and the result is really reproduceable, i.e. no typos etc.

Important is that the head node is configured and works appropriately. This is especially important for NIS/NFS/DNS stuff which can take a while to debug.

Furthermore, it is nice if one has access to IPMI enabled nodes. This makes debugging the start-up procedure not really more comfortable but at least somewhat bearable.

The script to build the live nfs root (bootstrap.sh) is hosted here on BitBucket. Usage is very simple:

```
cd debian-diskless-cluster
./bootstrap.sh -d /nfsroot/directory
```

Check out the `test.sh` script if you need to rebuild often:

1. It deletes the preset nfsroot
2. debotstrap into it

3. reboots a node via ipmi

Another helper script is `diskless-lib`. This provides mount_chroot_image and umount_chroot_image commands to un-/mount a chrooot and the `proc`, `sys` and `devpts` special directories. That script is adapted from kestrel-hpc.

THe `bootstrap.sh` script has been tested with sid (2013-12-10) as the target version and the node boots properly. The mismatch of the SLURM versions prevents usage in a mixed system, though.

2.2 Optional Setup: `eatmydata` (for faster debootstrapping)

Using `eatmydata` speeds up the build process quite a bit by ignoring the multiple `fsync()` etc. calls from dpkg/apt-get etc. This means that the data is *not yet* commited to hard disk, in case of a hard reset this unwritten data could be lost. Eventually, the kernel will write the data to disk.

This simple two line patch does the trick (found somewhere on the net, the patch is actually reversed):

```
# /usr/share/debootstrap/scripts/wheezy
--- sid  2013-02-15 11:03:15.384977238 -0500
+++ sid.orig     2013-02-15 10:50:23.381293976 -0500
@@ -16,7 +16,7 @@
 esac

work_out_debs () {
-       required="$(get_debs Priority: required) eatmydata"
+       required="$(get_debs Priority: required)"
                if doing_variant - || doing_variant fakechroot; then
                   #required="$required $(get_debs Priority: important)"
@@ -68,7 +68,7 @@
       second_stage_install () {
        setup_devices
-       export LD_PRELOAD=/usr/lib/libeatmydata/libeatmydata.so
+
         x_core_install () {
                   smallyes '' | in_target dpkg --force-depends --install
$(debfor "$@")
```

2.3 Important Preparations for Successful Network Boot

In order to succesfully boot from the network some configuration details have to be obeyed strictly. When setting this up for the first time it was really frustating because every configuration change requires a node reboot. Without ipmi and console redirection this becomes even more tedious. The following setup works and is in actual use.

2.3.1 Configure PXE bootparamter

PXE boot environment configuration is done with the file `/srv/tftp/pxelinux.cfg/default`. Here is an example `APPEND` line:

```
APPEND boot=aufs nfsroot=192.168.0.254:/srv/node-image ro initrd=initrd.img-3.2.0-4-amd64
```

CRUCIAL: Leave out the `ip=` kernel parameter!!!

2.3.2 Initramfs configuration

If the *DEVICE* parameter is left empty the *ipconfig* command of the kernel will request an IP address from all NICs. The second NIC is not connected and thus waits forever for an answer.

The `DEVICE=eth0` ensures that *only* `eth0` will request an IP on that device.

Paramters to change in `/srv/node-image/etc/initramfs-tools/initrams.conf`:

```
DEVICE=eth0
NFSROOT=auto
```

2.3.3 Overlay File System

Add `aufs` to the `/srv/node-image/etc/initramfs-tools/modules`. We will need this module to overlay the read only NFS root directory so that some important files can be written. The file will be *overlayed* over the original file:

```
echo "aufs" >> /srv/node-image/etc/initramfs-tools/modules
```

2.3.4 Initrd Script for Overlay File System

The kernel parameter `boot=aufs` in the PXE config above starts the script `aufs` in the folder `/srv/node-image/etc/initramfs-tools/scripts` (adapted from here):

```
#!/bin/sh
mountroot()
{
    ROOTTMPFSSIZE='500M';
    for x in $(cat /proc/cmdline); do
        case $x in
            roottmpfssize=*)
                ROOTTMPFSSIZE=${x#roottmpfssize=}
                echo "Root tmpfs size is $ROOTTMPFSSIZE"
                sleep 1
            ;;
        esac
    done

    modprobe nfs
    modprobe af_packet
    modprobe aufs
    udevadm trigger
    wait_for_udev 5
    configure_networking
    test -d /nfsroot || mkdir /nfsroot
    test -d /ramdisk || mkdir /ramdisk
```

```
23     test -d /${rootmnt} || mkdir ${rootmnt}
       sleep 3
25
       mount -t tmpfs -o rw,size=$ROOTTMPFSSIZE tmpfs /ramdisk
27     retry_nr=0
       max_retry=30
29     while [ ${retry_nr} -lt ${max_retry} ] && [ ! -e /nfsroot/${init} ]; do
           log_begin_msg "Trying nfs mount"
31         nfsmount -o nolock,ro  ${NFSOPTS} ${NFSROOT} /nfsroot
           /bin/sleep 1
33         retry_nr=$(( ${retry_nr} + 1 ))
           log_end_msg
35     done
       # overlay /ramdisk(rw) over /nfsroot(ro) and mount it on /
37     mount -t aufs -o dirs=/ramdisk=rw:/nfsroot=ro none ${rootmnt}
       echo ${hostname} > ${rootmnt}/etc/hostname
39     echo "cluster" > ${rootmnt}/etc/defaultdomain
       echo "live-node" > ${rootmnt}/etc/debian_chroot
41
   }
```

This script will first load a couple of kernel modules (lines 15-17):

1. `nfs` to be able to mount NFS volumes
2. `af_packet` allows the user to implement protocol modules in user space
3. `aufs`, our overlay file system driver

Afterwarsds it will wait for `udev`to populate the devices before it creates a `tmpfs` file system in RAM with 500 MBytes (lines 19-26).

It then goes on and tries to mount the NFS root directory until it succeeds. Upon success the ramdisk and the nfs root are *united* to the new root mount point. The ramdisk will be writeable and contains all the changes to the filesystem in memory until the next reboot. The nfsroot will be read-only.

2.3.5 Prevent reconfiguration of the node network interfaces

Make sure that the `$IMGDIR/etc/network/interfaces` file has this entry:

```
...
iface eth0 inet manual
....
```

This insures that the NIC configuration will be left as it is and not reconfigured, which would break the connection to the NFS root.

2.3.6 These directories should be created

The aufs script needs these directories to exist on the live image:

```
mkdir /srv/node-image/nfsroot /srv/node-image/ramdisk
```

2.3.7 Silence /etc/mtab errors

To prevent the error `/etc/mtab not found!` we simply link `/proc/mounts` to `/etc/mtab`:

```
chroot /srv/node-image ln -s /proc/mounts /etc/mtab
```

2.3.8 We also need the nfs-client software

Of course we need the NFS client to connect to our shared `/home` and `/data` directories:

```
apt-get install nfs-common
```

2.4 NIS

The NIS clients need to add the `+:::` and similar to `/etc/passwd`, `/etc/shadow`, and `/etc/group` files. This will make the clients ask the master server for user credentials like UID, GID, etc.

The file `/etc/nsswitch` should be changed slightly, change all occurances (for `passwd` etc.) from `compat` to `nis compat`.

Make the NIS client start on system start up (`/etc/default/nis`):

```
NISCLIENT=true
```

Then, the client needs to know *which* server to ask (`/etc/yp.conf`):

```
ypserver 192.168.0.254
```

That's it for the NIS client setup! If somthing is not working check first if DNS is working, then recreate the yp maps on the master node.

2.5 SLURM and Munge

CRUCIAL: The time for all hosts on the cluster has to be correct, otherwise munge will not work (use `ntpdate` for the first clock setup, then install `ntp` daemon to keep it synchronized)!

2.5.1 Munge

SLURM will use the `munge` daemon to issue commands securely on the nodes. The installation is very simple:

```
aptitude install munge
# create a new key
create-munge-key
```

The key file, `/etc/mung/munge.key` needs to be accesible and identical for every node. The permissions have to be set to 0600, otherwise `munge` won't start! Check in `/var/log/munge/munged.log` for errors, check with `pgrep munge` if the daemon is running.

Testing the `munge` installation is easy once a node is running:

```
root@test-master-node:~# echo "test" | munge
MUNGE:AwQDAADCE7qjEZ3xHGxnSQI/aZk16N+K35T+2vf3O3/YxHa6z31CzxZz9MAYXq9uZV8pBYrSY4VtatYbPtxIrx3Ke6Dgi/AIztl2JO3SABm+IyTk104bB8I=:

root@test-master-node:~# echo "test" |  munge | ssh linux-20 unmunge
STATUS:           Success (0)
ENCODE_HOST:      testserver.cluster-agvogel (10.0.0.161)
ENCODE_TIME:      2013-12-09 15:02:41 (1386597761)
DECODE_TIME:      2013-12-09 15:02:43 (1386597763)
TTL:              300
CIPHER:           aes128 (4)
MAC:              sha1 (3)
ZIP:              none (0)
UID:              root (0)
GID:              root (0)
LENGTH:           5

test
```

A common pitfall is to copy the key from the master to the nodes nfsroot. The UID and GID of the munge daemon on the master and nodes do not necessarily conform. Make sure that the owner of the file in the nfsroot is indeed the UID/GID of the nfsroot munge. You can check for the UID/GID and set the owner with:

```
chroot /srv/nfsroot id munge
chroot /srv/nfsroot chown munge:munge /etc/munge/munge.key
chroot /srv/nfsroot chmod 600 /etc/munge/munge.key
```

2.5.2 SLURM

There is a web configuration script in `/usr/share/doc/slurm-llnl` but it seems to be slightly outdated: gang" scheduling is no longer a seperate module, it is now builtin. In order for gang scheduling to work one must set the `preemptmode` appropiately (see below for an example `slurm.conf`:

```
preemptmode=GANG
```

Following are the current active settings:

```
root@test-master-node:~# cat /etc/slurm-llnl/slurm.conf |grep -v  '^$' | grep -v '=\s*$'|grep -v '^#'
ControlMachine=test-master-node
ControlAddr=192.168.0.254
AuthType=auth/munge
CacheGroups=0
CryptoType=crypto/munge
JobCheckpointDir=/var/lib/slurm-llnl/checkpoint
MpiDefault=none
ProctrackType=proctrack/pgid
ReturnToService=1
SlurmctldPidFile=/var/run/slurm-llnl/slurmctld.pid
SlurmctldPort=6817
```

```
SlurmdPidFile=/var/run/slurm-llnl/slurmd.pid
SlurmdPort=6818
SlurmdSpoolDir=/var/lib/slurm-llnl/slurmd
SlurmUser=slurm
StateSaveLocation=/var/lib/slurm-llnl/slurmctld
SwitchType=switch/none
TaskPlugin=task/none
InactiveLimit=0
KillWait=30
MinJobAge=300
SlurmctldTimeout=120
SlurmdTimeout=300
Waittime=0
FastSchedule=1
SchedulerTimeSlice=60
SchedulerType=sched/builtin
SchedulerPort=7321
SelectType=select/cons_res
SelectTypeParameters=CR_Core_Memory
PreemptMode=GANG
AccountingStorageType=accounting_storage/none
AccountingStoreJobComment=YES
ClusterName=cluster-agvogel
JobCompType=jobcomp/none
JobAcctGatherFrequency=30
JobAcctGatherType=jobacct_gather/none
SlurmctldDebug=3
SlurmctldLogFile=/var/log/slurm-llnl/slurmctld.log
SlurmdDebug=3
SlurmdLogFile=/var/log/slurm-llnl/slurmd.log
NodeName=linux-[01-04] RealMemory=8000 Sockets=2 CoresPerSocket=4 ThreadsPerCore=1 State=UNKNOWN
NodeName=linux-[05-20] RealMemory=16000 Sockets=2 CoresPerSocket=12 ThreadsPerCore=1 State=UNKNOWN
PartitionName=nodes Nodes=linux-[01-20] Default=YES MaxTime=INFINITE State=UP Shared=FORCED:2
root@test-master-node:~#
```

2.6 RSYSLOG

This will configure rsyslog to send all messages to the head node (complete file without comments and empty lines):

```
root@test-master-node:~# grep -v -e ^# -e "^$" /srv/test-bootstrap/etc/rsyslog.conf
$ModLoad imuxsock # provides support for local system logging
$ModLoad imklog   # provides kernel logging support
$ModLoad immark  # provides --MARK-- message capability
$ActionFileDefaultTemplate RSYSLOG_ForwardFileFormat
$FileOwner root
$FileGroup adm
$FileCreateMode 0640
$DirCreateMode 0755
```

```
$Umask 0022
$WorkDirectory /var/spool/rsyslog
$IncludeConfig /etc/rsyslog.d/*.conf
*.* @@192.168.0.254
```

2.7 Mounted file systems

The filesystems needed on the nodes shoudl be defined in the file `/etc/fstab`. Here is an example (do not change the first three entries):

```
# /etc/fstab: static file system information.
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
proc     /proc       proc     defaults    0 0
sysfs    /sys        sysfs    defaults    0 0
devpts   /dev/pts    devpts   rw,nosuid,noexec,relatime,gid=5,mode=620     0 0
192.168.0.254:/home /home nfs4 defaults 0 0
192.168.0.200:/data /data nfs4 defaults 0 0
```

2.8 Prevent Starting of Daemons

The following command will prevent the start of daemons/services during installation as we do *not* want to start a second ssh daemon in the nfs root chroot. More details of how this actually works can be found here, and in the man pages of invoke-rc.d, `man invoke-rc.d`. Use the following two commands to create a file and set the mode to be excutable:

```
echo -e "#!/bin/sh\necho Not starting daemon\nexit 101"  \
> $IMAGEDIR/usr/sbin/policy-rc.d
chmod 755 $IMAGEDIR/usr/sbin/policy-rc.d
```

---

3 Troubleshooting

Here are some useful commands for troubleshooting. Parameters starting with $ are variables you have to change to your needs.

```
cd /var/yp; make all # recreate yp maps
dig +short $hostname # check DNS, returns IP of $hostname
dig  @$dns_Server_ip +short $hostname # check DNS, using $dns_server_ip server
ypcat passwd.byname # check NIS on client/server, returns part of /etc/passwd
lynis # check system for obvious configuration oversights (security wise)
ifconfig eth0 192.168.0.1 # set IP of NIC eth0 to 192.168.0.1
mpirun -n 4 -H linux-20,localhost $cmd  # execute $cmd  linux-20 and locally
dsh -a -c -m $cmd # exec. $cmd on all nodes concurrently, prepend node name  to output
usermod -R $IMAGEDIR      # -R give the chroot envirnmoent for usermod,pwck, and friends
```

---

4 Useful Packages

Here is a list of useful packages with a short explanation of what they do:

- `etckeeper` manages the `/etc` directory. I use the following options: `AVOID_DAILY_AUTOCOMMITS=1` `AVOID_COMMIT_BEFORE_INSTALL=1`
- `lynis` is a auditing tool, helps to find obvious security problems.
- `schroot` makes it easy to maintain `chroot` environments.
- `dsh` executes commands remotely on several different machines at the same time.
- `vim` is my favorite editor for config files, YMMV.

---

5 Miscellaneous Stuff

5.1 Infinfiband: Setup and Testing

I wanted to use InfiniBand and bought two Mellanox ConnectX 20 GB/s QDR NICs. The idea was to use one node exclusively for data analysis which needs fast I/O, i.e. high bandwidth as well as low latency. InfiniBand is optimal for this cases, especiially for low latency. The fastest way to access NFS via InfiniBand is to use RDMA (*R*emote *D*irect *M*emory *A*ccess). In our case I decided for simplicity to use IP-over-Infiniband und use regular NFS with TCP/IP. A short instruction on how to actually make our InfiniBand work is following. Background information about InfiniBand can be found in the Wikipedia article and the references therein. A very good HowTo is offered from inqbus.

5.1.1 Physical Installation

The cards have a PCIe 2.0 8x interfcce. Open the computer and select a slot where they can fit in. In our NAS the lowest port is only PCIe2.0 x4 and slows down the card unnecessarily! You can check with `lspci` for the installed card:

```
root@test-master-node:~# lspci

04:00.0 InfiniBand: Mellanox Technologies MT26418 [ConnectX VPI PCIe 2.0 5
    ↪ GT/s - IB DDR / 10GigE] (rev a0)
```

The card is installed now, lets look at it more thoroughly with `lspci -vv`:

```
root@test-master-node:~# lspci -s 04:00.0 -vv
04:00.0 InfiniBand: Mellanox Technologies MT26418 [ConnectX VPI PCIe 2.0 5
    ↪ GT/s - IB DDR / 10GigE] (rev a0)
    Subsystem: Mellanox Technologies Device 0001
    Control: I/O- Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr-
        ↪ Stepping- SERR+ FastB2B- DisINTx+
```

```
    Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort- <TAbort-
        ↪ <MAbort- >SERR- <PERR- INTx-
    Latency: 0, Cache Line Size: 64 bytes
    Interrupt: pin A routed to IRQ 19
    Region 0: Memory at f4d00000 (64-bit, non-prefetchable) [size=1M]
    Region 2: Memory at fd000000 (64-bit, prefetchable) [size=8M]
    Capabilities: [40] Power Management version 3
        Flags: PMEClk- DSI- D1- D2- AuxCurrent=0mA PME(D0-,D1-,D2-,D3hot-,
            ↪ D3cold-)
        Status: D0 NoSoftRst- PME-Enable- DSel=0 DScale=0 PME-
    Capabilities: [48] Vital Product Data
        Product Name: Eagle DDR
        Read-only fields:
            [PN] Part number: MHGH19-XTC
            [EC] Engineering changes: A1
            [SN] Serial number: MT1045X00466
            [V0] Vendor specific: PCIe Gen2 x8
            [RV] Reserved: checksum good, 0 byte(s) reserved
        Read/write fields:
            [V1] Vendor specific: N/A
            [YA] Asset tag: N/A
            [RW] Read-write area: 111 byte(s) free
        End
    Capabilities: [9c] MSI-X: Enable+ Count=256 Masked-
        Vector table: BAR=0 offset=0007c000
        PBA: BAR=0 offset=0007d000
    Capabilities: [60] Express (v2) Endpoint, MSI 00
        DevCap: MaxPayload 256 bytes, PhantFunc 0, Latency L0s <64ns, L1
            ↪ unlimited
            ExtTag- AttnBtn- AttnInd- PwrInd- RBE+ FLReset-
        DevCtl: Report errors: Correctable- Non-Fatal- Fatal- Unsupported-
            RlxdOrd- ExtTag- PhantFunc- AuxPwr- NoSnoop-
            MaxPayload 128 bytes, MaxReadReq 512 bytes
        DevSta: CorrErr- UncorrErr- FatalErr- UnsuppReq- AuxPwr- TransPend-
        LnkCap: Port #8, Speed 5GT/s, Width x8, ASPM L0s, Latency L0
            ↪ unlimited, L1 unlimited
            ClockPM- Surprise- LLActRep- BwNot-
        LnkCtl: ASPM Disabled; RCB 64 bytes Disabled- Retrain- CommClk-
            ExtSynch- ClockPM- AutWidDis- BWInt- AutBWInt-
        LnkSta: Speed 2.5GT/s, Width x8, TrErr- Train- SlotClk- DLActive-
            ↪ BWMgmt- ABWMgmt-
        DevCap2: Completion Timeout: Range ABCD, TimeoutDis+
        DevCtl2: Completion Timeout: 50us to 50ms, TimeoutDis-
        LnkCtl2: Target Link Speed: 5GT/s, EnterCompliance- SpeedDis-,
            ↪ Selectable De-emphasis: -6dB
             Transmit Margin: Normal Operating Range,
                 ↪ EnterModifiedCompliance- ComplianceSOS-
             Compliance De-emphasis: -6dB
        LnkSta2: Current De-emphasis Level: -6dB, EqualizationComplete-,
            ↪ EqualizationPhase1-
             EqualizationPhase2-, EqualizationPhase3-,
                 ↪ LinkEqualizationRequest-
    Capabilities: [100 v1] Alternative Routing-ID Interpretation (ARI)
        ARICap: MFVC- ACS-, Next Function: 1
```

```
        ARICtl: MFVC- ACS-, Function Group: 0
    Kernel driver in use: mlx4_core
```

The important lines are the ones beginning with `LnkCap` and `LnkSta`, which tell us the cards capability and current link status:

```
root@test-master-node:~# lspci -s 04:00.0 -vv|grep -e LnkSta: -e LnkCap:
        LnkCap: Port #8, Speed 5GT/s, Width x8, ASPM L0s, Latency L0
            ↪ unlimited, L1 unlimited
        LnkSta: Speed 2.5GT/s, Width x8, TrErr- Train- SlotClk- DLActive-
            ↪ BWMgmt- ABWMgmt-
```

This means the card does not get maximum speed (Speed 5GT/s, Width x8) but only half of the maximum (Speed 2.5GT/s, Width x8). This mainboard is only capable of PCIe 1.0 (2.5 GTransfers/s) and the card will not reach the theoretical bandwitdh of 20 GB/s but only half of that. The data rate is given *per direction* which is still impressive. For our use case the latency is more important, anyway.

5.1.2 Necessary Software

We need to install drivers and several network test packages:

```
aptitude install perftest ibutils libmlx4-1 opensm
```

In order to make our InfiniBand cards able to talk to each other there need to be a so called *subnet manager*. One such SM is included in the package `opensm`.

We can load the driver for our IB cards with `modprobe mlx4_core` if it is not loaded yet. Confirm succesfull detectionf of the card with `dmesg |grep Mellanox`. The card's status can be viewed with `ibstatus` command:

```
root@testserver:~# ibstatus
Infiniband device 'mlx4_0' port 1 status:
        default gid:     fe80:0000:0000:0000:0002:c903:000d:b717
        base lid:        0x2
        sm lid:          0x1
        state:           4: ACTIVE
        phys state:      5: LinkUp
        rate:            20 Gb/sec (4X DDR)
```

Upon installing `opensm` the cards state should change to *ACTIVE*. If you need the `port` of your card for `opensm`, you can use the command `ibstat -p` and add the port in `/etc/default/opensm`:

```
root@test-master-node:~# ibstat -p
0x0002c903000db717
```

Now, we need to load the IP-over-Infiniband kernel modules. They are not loaded automatically so do not forget to add them to `/etc/modules`:

```
mlx4_ib
ib_ipoib
ib_umad
rdma_ucm
rdma_cm
```

Load them with `modprobe`. Then we can finally assign an IP to our new card:

```
ifconfig ib0 up 192.168.2.1
```

Following this instruction on the second host should allow you to ping the cards. Create a new entry in `/etc/network/interfaces` to make the assigned IP permanent:

```
auto ib0
iface ib0 inet static
    address 192.168.1.1
    netmask 255.255.255.0
```

5.1.3 InfiniBand Performance

You can use `iperf` to test the IP network bandwidth, as well as `ib_read_lat`, `ib_write_bw`, etc. to test the read latency or the write bandwidth, respectively.

Here are some results from `iperf`:

```
root@testserver:~# iperf -c 192.168.1.2 -P4
------------------------------------------------------------
Client connecting to 192.168.1.2, TCP port 5001
TCP window size:  649 KByte (default)
------------------------------------------------------------
[  5] local 192.168.1.1 port 56344 connected with 192.168.1.2 port 5001
[  4] local 192.168.1.1 port 56341 connected with 192.168.1.2 port 5001
[  3] local 192.168.1.1 port 56342 connected with 192.168.1.2 port 5001
[  6] local 192.168.1.1 port 56343 connected with 192.168.1.2 port 5001
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0- 9.0 sec  1.40 GBytes  1.34 Gbits/sec
[  3]  0.0- 9.0 sec  1.42 GBytes  1.35 Gbits/sec
[  6]  0.0- 9.0 sec  1.41 GBytes  1.34 Gbits/sec
[  5]  0.0-10.0 sec  2.04 GBytes  1.75 Gbits/sec
[SUM]  0.0-10.0 sec  6.27 GBytes  5.39 Gbits/sec
```

For comparison look at the speed of 1G ethernet:

```
root@testserver:~# iperf -c 10.0.0.102 -P 4
------------------------------------------------------------
Client connecting to 10.0.0.102, TCP port 5001
TCP window size: 23.5 KByte (default)
------------------------------------------------------------
[  5] local 10.0.0.161 port 41893 connected with 10.0.0.102 port 5001
[  3] local 10.0.0.161 port 41890 connected with 10.0.0.102 port 5001
[  4] local 10.0.0.161 port 41891 connected with 10.0.0.102 port 5001
[  6] local 10.0.0.161 port 41892 connected with 10.0.0.102 port 5001
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0-10.0 sec   188 MBytes   157 Mbits/sec
[  3]  0.0-10.0 sec   203 MBytes   170 Mbits/sec
[  4]  0.0-10.0 sec   174 MBytes   146 Mbits/sec
[  6]  0.0-10.3 sec   200 MBytes   164 Mbits/sec
[SUM]  0.0-10.3 sec   766 MBytes   627 Mbits/sec
```

It is usually around 1000 MBit/s per second, the reason for the discrepancy is maybe that the NAS was not idle during this test.

Here are the results for `ib_write_bw` and `ib_write_lat`:

```
root@nas-2:~# ib_write_bw 192.168.1.1
------------------------------------------------------------------
                    RDMA_Write BW Test
Number of qp's running 1
Connection type : RC
Each Qp will post up to 100 messages each time
Inline data is used up to 1 bytes message
  local address:  LID 0x01, QPN 0x004b, PSN 0x2984f8 RKey 0x10001b00 VAddr
    ↪ 0x007fbee52f7000
  remote address: LID 0x02, QPN 0x2a004b, PSN 0x97f807, RKey 0xa6001b00
    ↪ VAddr 0x007f5d2ba1f000
Mtu : 2048
------------------------------------------------------------------
 #bytes #iterations    BW peak[MB/sec]    BW average[MB/sec]
  65536          5000                1497.12                 1497.03
------------------------------------------------------------------
```

```
root@nas-2:~# ib_write_lat 192.168.1.1
------------------------------------------------------------------
                    RDMA_Write Latency Test
Inline data is used up to 400 bytes message
Connection type : RC
   local address: LID 0x01 QPN 0x2004b PSN 0xe85578 RKey 0x12001b00 VAddr 0
     ↪ x000000023a0002
  remote address: LID 0x02 QPN 0x2c004b PSN 0x916bac RKey 0xa8001b00 VAddr
    ↪ 0x00000000962002
Mtu : 2048
------------------------------------------------------------------
 #bytes #iterations    t_min[usec]    t_max[usec]  t_typical[usec]
      2          1000            1.30          66.57             1.33
------------------------------------------------------------------
```

5.1.4 Tuning

You can change the connection mode from `datagram` to `connected` which will allow for MTU sizes up to 65520 bytes instead of 2044 bytes but drops mutlicast packets. This settings change is accomplished via the `/sys` virtual file system:

```
echo "connected" > /sys/class/net/ib0/mode
ifconfig ib0 mtu 65520
```

Further information about InfiniBand and tuning can be found in the Mellanox OFED for Linux User's Manual(PDF) and Performance Tuning Guide for Mellanox Network Adapters(PDF).

5.1.5 NFS over IP-over-IB

Using iozone to check the performance of the InfiniBand mounted NFS share we get following performance statistics:

You can see that once the file size is around/bigger than the RAM size (22 GB in this case) the write speed is consistently at around 500 MBytes/s. This corresponds to the

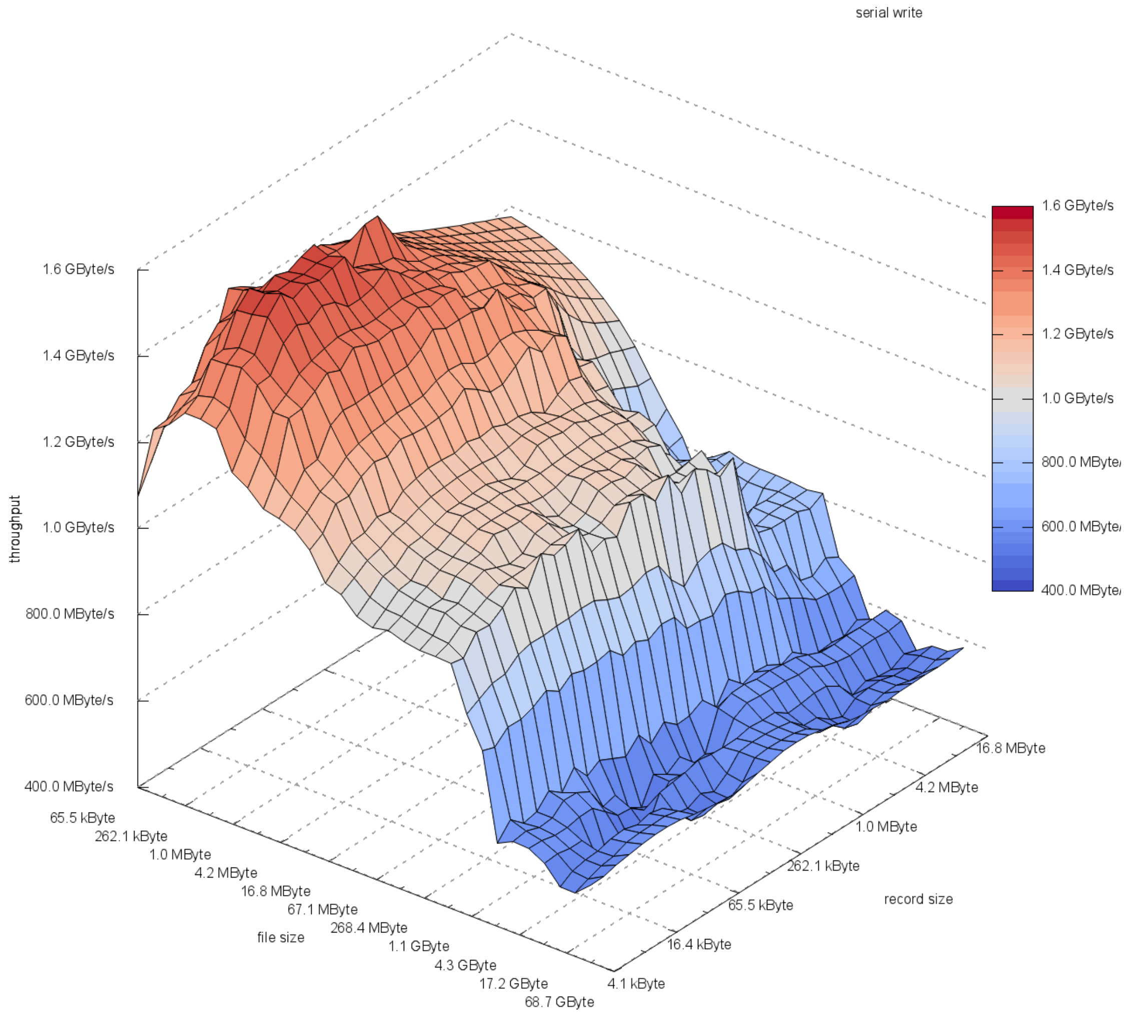

Figure 2: IB performance statistics: via IB mounted NFS share from a 12 Disk RAID6 NAS

local speed (on the NAS) of about 600 MBytes/sec. At smaller file sizes the caching of the server is the only limitting factor (CPU cache and RAM speed, which are much faster than disks). According to various sources on the net, one could expect an improvement of about 10% to 20% by usage of so called RDMA-NFS (*R*emote *D*irect *M*emmory *A*ccess). We are already quite close to the local throughput, so I do not think it is worth the complication for our case. I found no numbers about the latency differences, but i would expect that RDMA-NFS performes better.

5.1.6 BTW!

- nmap fails on wheezy due to the infiniband card

5.2 MRTG Setup on master-node

Here is a shsort installation note about MRTG. This little program collects sttatistics about the traffic from each port of the switch via SNMP. Do not forget to enable SNMP on the switch or this will not work! This can be done either with the routers web-interface or by ssh if the switch supports it. The following commands enable SNMP acces from the master-node on an SMC TigerSwtich:

```
ssh admin@router
configure
management snmp-client $master-node-ip
exit
copy running-config startup-config
```

Then install `mrtg`:

```
aptitude install mrtg mrtgutils --with-recommends
mkdir /var/ww/mrtg
chown www-data:www-data /var/www/mrtg
# create config
cfgmaker public@router-ip > /etc/mrtg-router.cfg
# run mrtg once, to create initital rrds and graphs
env LANG=C /usr/bin/mrtg /etc/mrtg-router.cfg
# create html index file
indexmaker /etc/mrtg-router.cfg > /var/www/mrtg/index.html```

Add the following line to your crontab (with ```crontab -e```) to update the statistics every 5 minutes:
```

/5 * * * env LANG=C /usr/bin/mrtg /etc/mrtg-smc-switch.cluster.cfg > /dev/null

```

The output should be seen with browser on ```http://host/mrtg```.

## Postfix Mail steup on master-node

Goals:

* Accept emails from the private network, so we can send status mails from various hosts and services, like from a NAS or RAID Managmant Software.
```

```
* Relay mails to our own mailserver (external_mail_host), and *only* to our mailserver with domain "external_domain"


This setup can be done in the follong way with the postfix MTA:

1.  Edit the ```/etc/postfix.conf``` file:

    ```
    transport_maps = hash:/etc/postfix/transport
    mydestination = external_hostname, master-node, master-node.cluster, localhost
    mynetworks = 127.0.0.0/8 [::ffff:127.0.0.0]/104 [::1]/128 192.168.0.0/24
    smtp_generic_maps = hash:/etc/postfix/generic
    ```
2. Create the file ```/etc/postfix/transport``` with the following content (don't forget to `postmap` it):

    ```
    external_domain smtp:external_mail_host
    external_fqdn :
     .cluster :
    * error:"relay to other domains forbidden"
    ```
3. Create the file ```/etc/postfix/generic``` with the following content (don't forget to `postmap` it):

    ```
    @nas1.domain nas1@external_domain
    @nas2.domain nas2@external_domain
    ```

Restart postfix. Emails from nas1 will be relayed to external_domain. The external_mail_host only accepts mail from resolvable hostnames, so we have to rewrite the from address (the rules are in /etc/postfix/transport). In order to check the configuration try to send some mails from inside to master-node, some external mail address, etc. Check in ```/var/log/mail.log``` and check the mail queue with ```mailq``` command.
You can delete queued mails with ```post super -D ALL```.

GlusterFS

Our Problem: We need more disk space, but we have no money for another NAS.
Possible Solution: Put a disk in each node and combine them all into to one big volume using glusterfs

Unfortunately we need to store information for the gluster daemon *per node*.

Prepare Semi-stateful nodes

```

```
Yes, this is hilarous: we go through hoops to create a diskless/stateless
    ↪ cluster only to go back and put disks in it.
We create a script which we will run after the system booted (with /etc/rc.
    ↪ local). Following steps are needed befor starting glusterfs−server:

1. Mount the glusterfs relevant directories via NFS
2. Mount the bricks, create proper mount points
3. Only then start the glusterfs dameon
```

6 !/bin/bash

echo “Mounting glusterfs related directories” mkdir -p /var/lib/glusterd mount -t nfs4 nas1:/stateful/$HOSTNAME/var/lib/glusterd/var/lib/glusterdmkdir-p/var/log/glusterfsmount-tnfs4nas1:/stateful/${HOSTNAME}/var/log/glusterfs /var/log/glusterfs # XFSDEVS=($blkid-odevice-tTYPE=xfs|sort)echo"Foundxfsdevices$:XFSDEVS” BRICK_NUM=1 for dev in $XFSDEVS;doecho"Mountingbrick$:dev to /gluster/brick$BRICK_NUM"mkdir-p/gluster/brick${BRICK_NUM} mount /dev/sda1 /gluster/brick$BRICK_NUMvar_i$ =((BRICK_NUM + 1)) done # if [$(pgrep gluster)]; then /etc/init.d/glusterfs-server restart else /etc/init.d/glusterfs-server start fi “‘ ## GlusterFS Volume Configuration

If you have different sized bricks you should set a cluster.min-free-size quota to a specific value and not 5%. `gluster volume set <volname> cluster.min-free-disk 30GB`

Best is to avoid uneven brcik sizes, as that is not tested much. Furthermore is the algorithm used to spread the files not meant for that usecase.

If you see following error in the logs:

```
0−management: connection attempt failed (Connection refused)
```

check if the UUID of all hosts match, if not detach the offending host and probe again.

7 Printer Setup

We do not (yet) have our own subnet, we do not (yet) use a configuration managment system like puppet, salt, or cfgengine, so we have to manage some stuff via command line. Like setting up a printer:

```
dsh −c −M −a lpadmin −p colorqube −E  −v ipp://130.83.32.235/ipp −P ./
    ↪ xr_ColorQube8570.ppd  −o printer−is−shared=false −o PageSize=A4 −o
    ↪ XRXOptionFeatureSet=DN
dsh −c −M −a lpoptions −d colorqube
```